SETTING UP FOR THE PROJECT

Step 1: Create the database:

1) Open an EXISTING database connection in DbVisualizer.
2) Run the command `CREATE DATABASE birthdaybook;`
3) Close the existing connection.
4) Create a DbVisualizer connection for the birthdaybook database and connect to it.
5) Once your NEW connection is open, open the file
   `sample-birthday-book-project\database\birthdaybook.sql`
6) Execute the script to create the database and a few records;

*****NOTE: The database name MUST be birthdaybook in order for tests to work correctly*****

Step 2: Import the project into Eclipse.

Step 3: Before you start coding, confirm your setup is correct by:

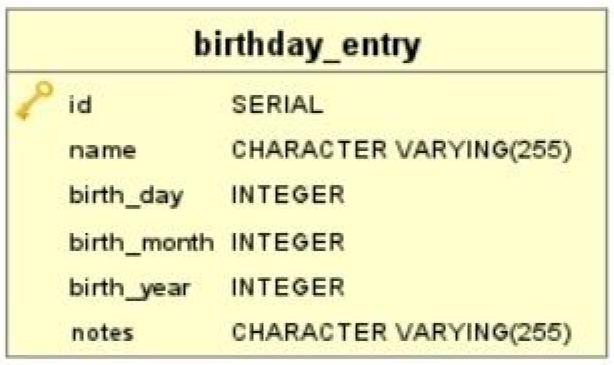
1) Open the file `JDBCBirthdayEntryDAOTest` class in the `com.techelevator.birthdaybook.dao package` of the `src/test/java folder` .
2) Run the test `create_withValidData_shouldCreateRecord`
3) All the other tests in this class will fail but if your setup is correct this one will pass.

Step 4: Your setup is COMPLETE!!!!!

PART 1: The DAO Code

This project is a “birthday book” - basically an app that lets you store information about people and their birthdays. The data model is a bit contrived but it’s done this way to allow for practice of various topics you have learned.

The data model looks like this:



birthday_entry	
id	SERIAL
name	CHARACTER VARYING(255)
birth_day	INTEGER
birth_month	INTEGER
birth_year	INTEGER
notes	CHARACTER VARYING(255)

Step 1: Start with completing the database code:

The **`JDBCBirthdayEntryDao`** class in the **`com.techelevator.birthdaybook.dao`** package has been provided for you.

Currently, the **`create`** method is completed… it is up to you to fill in the rest of the methods:

1) Start with the **`getEntries()`** method:
    a) Write the code to get all the **`BirthdayEntry`** objects in the database.
    b) The method **`mapRowToBirthdayEntry`** to map a **`SqlRowSet`** row to a **`BirthdayEntry`** has been provided but you will need to complete it in order to be able to use it.
    c) If you complete this correctly, the **`getEntries_withValidData_shouldReturnMultipleRecordsin`** the **`JDBCBirthdayEntryDAOTest`** class should pass.

2) Next, complete the **`getEntry(Long id)`** method:
    a) Write the code to get a specific **`BirthdayEntry`** record by id.
        i) **`getEntry_withValidId_shouldReturnRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.
    b) In the case that the record is not found, throw an **`EntryNotFoundException`** (this class has been provided but note that you will need to make some changes to the method signature in order to accomplish this).
        i) **`getEntry_withValidId_shouldReturnRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.

3) Next, complete the **`deleteEntry(Long id)`** method.
    a) Write the code to delete a specific BirthdayEntry record by id.
        i) **`deleteEntry_withValidId_shouldDeleteRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.
    b) In the case that the record is not found, throw an **`EntryNotFoundException`**.
        i) In order to be able to know whether the record was deleted or not, you will need to check the number of records affected. The **`jdbcTemplate.update`** method returns a count of affected rows as an integer (we haven't used this value so far but it does actually return a count and you can assign the return value to an int variable) and if that count is 0, the record did not exist.
        ii) **`deleteEntry_withInValidId_shouldThrowExceptionin`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.
        iii) **`deleteEntry_withValidId_shouldNotThrowException`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.

4) Next, complete the **`updateEntry(BirthdayEntry entry, Long id)`** method.
    a) Write the code to update a specific **`BirthdayEntry`** record by id.
        i) The basic skeleton has been provided. Your query should go where the comment **`// update here`** is.
        ii) **`updateEntry_withValidData_shouldUpdateRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.
    b) If the record does not exist, a **`DataAccessException`** will be thrown by the system. The provided code catches this exception but when it does, you should throw a **`EntryNotFoundException`** (this essentially changes the exception to beone related to your code rather than the systemone).
        i) **`updateEntry_withInvalidData__shouldThrowEntryNotFoundException`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.

c) The skeleton code checks to make sure that the id in the **`BirthdayEntry`** and the id provided match. If the scenario where the ids don't match occurs, you should also throw a **`EntryNotFoundException`**.
   i) **`updateEntry_withInvalidRecord_shouldThrowEntryNotFoundException`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.

At this point, all the tests in **`JDBCBirthdayEntryDAOTest`** should pass.

Congrats… you have finished Part 1!